TUNING OF A REINFORCEMENT LEARNING BASED CONTROLLER FOR A FOURTH ORDER FLUID LEVEL SYSTEM

L., MATOS G.

Laboratório de Automação e Robótica, Departamento de Engenharia Elétrica, Universidade de Brasília
L3 Norte, SG-11 – UnB Área 1, DF, 70919-970
E-mails: lguilhem.matos@gmail.com

A.,BAUCHSPIESS

Laboratório de Automação e Robótica, Departamento de Engenharia Elétrica, Universidade de Brasília
Faculdade de Tecnologica, Departamento de Engenharia Elétrica, UnB, DF, 70910-900

Abstract— This work presents the tuning of an adaptive controller using reinforcement learning and neural networks in order to deal with black-box time-variant nonlinear systems. To evaluate the controller's limitations, a fourth-order fluid level system was chosen because of its wide range of time constants and non-linearities. Implementation was made on a computer running MatLab® connected to an Arduino as interface to the sensor and actuator. The controller was tested with different sample times and different learning rates and, afterwards, was compared to a PI controller. The controller was able to perform inside specific learning and sample rate margins and if given time, shows adaptive and optimizing features that causes it to perform better than the PI.

Keywords— Reinforcement Learning, Neural Networks, Adaptive Control, Fluid Level System

Resumo— Este trabalho apresente o processo de ajuste de um controlador adaptativo com aprendizado por reforço e redes neurais utilizado para lidar com sistemas caixa-preta e variantes no tempo. Para avaliar as liitações do controlador um sistema de nível de líquidos de quatro tanques foi escolhido por possuir grande variedade de constantes de tempo, não linearidades e atraso considerável. A implementação foi feita em MatLab® utilizando-se um Arduino como interface entre o computador e o sensor e o atuador. O controlle foi submetido a testes de variação de taxa de amostragem e de aprendizado e em seguida foi comparado com um controlador PI. O controlador desenvolvido atuou de maneira estável se dentro de faizes definidas de taxas de aprendizado e amostragem e dado tempo suficiente demonstra características adaptativas e de otimização demonstrando performance superior à do PI.

Palavras-chave— Aprendizado por Reforço, Redes Neurais, Controle Adaptativo, Sistema de Nível de Líquidos.

1 Introduction

Time-variant systems are usual and can be found in several day-to-day applications. An aircraft that changes weight with fuel consumption is an example. Also, information on system dynamics is not always available, thus, solutions for this sort of systems should be studied and developed.

One way of dealing with black box time-variant system is to use adaptive control. The main goal of Adaptive controllers is to deal with lack of knowledge on the system being controlled and with parameter variation. This is time variance. This work's goal is to develop an adaptive controller that can be used on real systems dealing with its limitations and difficulties.

Reinforcement Learning is, in its essence, adaptive. It is a form of learning based on trial and error. Also, as pointed by Sutton et al. (1991) and Vrabie et al. (2014), reinforcement learning has its mathematical roots in optimal control. This means that Reinforcement Learning is an adaptive algorithm that also has optimizing features, even though it does not need information on the system to perform.

Reinforcement Learning usage for adaptive control dates back to the 80's, but works from Cui et al. (2017), Hwangbo et al. (2017), Lilicrap et al. (2016), Noel and Pandian (2014), Papierok et al. (2008), Ramanathan et al. (2018) and Rao et al. (2018) show that it is still a trend. Such works show that there is yet much to be developed.

As detailed in Section 2, estimating some functions is needed to use reinforcement learning and some of its methods. This work uses neural networks to approximate these functions. One of the main goals of neural networks lies in function approximation and in this case, they are particularly useful since no information on the system is available as described by Aguirre (2015). A multilayer perceptron was used to estimate the system dynamics in order to predict future outputs and bypass delay and radial basis neural networks were used to approximate the reinforcement learning functions. The estimation of functions with neural networks can be seen in work by Esfe (2017), Ferrari and Stengel (2005) and Yu et al. (2014).

A four-tank fluid level system was chosen as the experimental environment. The system has challenging dynamic features, such as great range of time constants, non-linearities and delay, besides, although often for more simple applications, fluid level control is

widely performed in industrial environment, thus, such systems could benefit from this study.

Results show that the controller, when inside specific margins for sample and learning rates, can perform stably. Regarding adaptivity, the controller can perform with no information whatsoever on the system, apart from the average delay. The controller shows clear signs of learning and optimizing features. It enhances performance over time lowering overshoot and time of oscillation in the vicinity of the reference value. When compared to the chosen PI, the controller performs poorly at the beginning, but, given time, its performance surpasses that of the PI.

In section 2 there is information on the concepts of reinforcement learning, the actor-critic method and the SARSA algorithm that is needed and sufficient to understand the following implementation. Section 3 presents the experimental environment and implementation details and section 4 the controller structure and the development of the algorithm used. Section 5 presents and analyzes the obtained results and finally, section 6 contains the conclusions and propositions for future work.

2 Reinforcement Learning

2.1 Concept and Elements

The Reinforcement Learning (RL) problem is formulated as an *Agent* or Individual interacting with a specific *environment* through *actions* defined according to an *action policy*. The environment is represented by *states*, which, for the RL problem, are the set of information available to the agent in a given moment. The actions chosen by the agent (through the action policy) will depend on such information.

By definition the RL state must have the Markovian Property which states that the information given in current time must contain all previous relevant information. This means the agent must be able to define an action to take using only current information.

Formally, the Markov Property states that the complete probability distribution for reaching a state *s* and reward *r*, given by

$$\mathbf{Pr}\{r_{t+1} = r, s_{t+1} = s | s_0, a_0, r_0, \dots, s_t, a_t, r_t\} \quad (1)$$

where s_t, a_t *and* r_t are respectively the state, the action taken and the reward received in time *t*, can be written as

$$\mathbf{Pr}\{r_{t+1} = r, s_{t+1} = s \mid s_t, a_t, r_t\}. \quad (2)$$

The environment's response to the action is called *reward* and it defines the immediate quality of an action. Usually it shows somehow if the agent got closer or further from its goal. Although high value rewards are accounted for, they represent an immediate response to the agent's actions, but the objective of the RL problem is to achieve maximum accumulated reward over time. This means the RL focus in achieve greater expected rewards from given state s_t. This is called the state's *value* or the *obtainable return* from that state defined as follows:

$$V(s_t) = \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} \quad (3)$$

which is the Bellman Equation of the reward with γ being the discount factor. γ determines that rewards that are obtained in further states are less important than those obtained in closer states. This equation defines how good it is for the agent to visit a given state in a long-term sense showing the reward expectation for the current state

Expansion of (3) gives:

$$V(s_t) = r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i} r_{t+i+1} \quad (4)$$

that can be written as,

$$V(s_t) = r_{t+1} + \gamma V(s_{t+1}) \quad (5)$$

which states that the sum of the discounted estimate of the next state value and the next state reward is the target to the actual estimate, as seen in Sutton and Barto (1998).

In practice the values of the states are not the real ones, but *estimates*. The difference between the target and the estimate of the value function is known as the Temporal Difference Error (δ_{td}), given by:

$$\delta_{td} = r_{t+1} + \gamma \bar{V}(s_{t+1}) - \bar{V}(s_t). \quad (6)$$

The δ_{td} will be used as parameter for updating the value function and the action policy function which are approximated by neural networks. Also, the objective is to minimize δ_{td} over time as the cost function of the RL problem.

2.2 The Actor-Critic Method

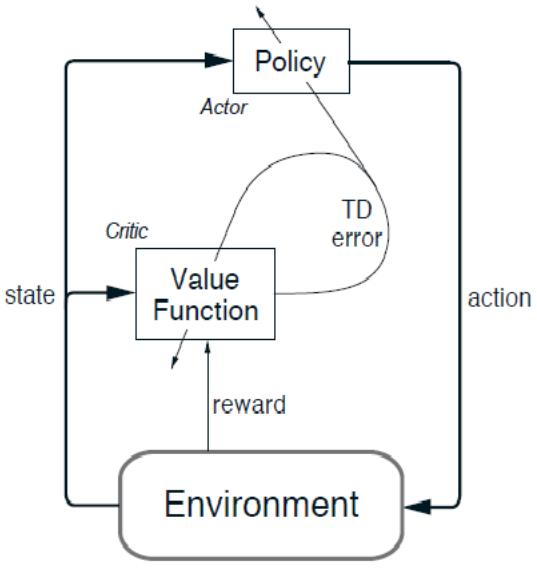

Figure 1. The Actor-Critic Structure

The Actor-Critic is a Temporal Difference (TD) method that has separated structures for the action policy, known as the Actor, and the value function estimator, known as the Critic. The Actor is the structure that defines which actions will be taken. In the control problem studied in this work it is the neural networks that generates the control signal. The critic is the structure that evaluates the actor's decisions. In this specific case, it bootstraps the value function using the reward to generate the δ_{td} which will be responsible for updating both neural networks. This procedure is depicted in Figure 1.

2.3 SARSA Algorithm

SARSA stands for 'state-action-reward-state-action' and is an TD algorithm. SARSA states that the state's value depends not only in the state variables, but also in the action chosen in said state. This means the estimate is the sum of expected future rewards after taking a specific action in a given state. Instead of using the function $V(s_k)$, the Algorithm uses the function $Q(s_k, a_k)$ as the value function for the Critic. In this case, the δ_{td} defined in (4) becomes

$$\delta_{td} = r_{t+1} + \gamma \bar{Q}(s_{t+1}, a_{t+1}) - \bar{Q}(s_t, a_t). \quad (7)$$

This is needed because the action taken will be also used to calculate the reward so it must be included as an input to the value function being approximated, otherwise, part of the reward's behavior will be uncorrelated to the value function, and so, the estimation would most likely fail.

3 Experimental Environment

To experiment on the controller's performance a fourth-order fluid level system with sequenced tanks and regulable fluid passage between each pair of tank was chosen. The fluid input is located on the bottom of the first, or far left tank illustrated as H1 in Figure 2 and the controlled level is the level of the fourth, or the far right tank illustrated as H4 in Figure 2. The connection between any two tanks is made by a regulable gate valve and the level measurement is made with a level sensor based on pressure with a tube to the bottom of the tank. The Set-Up uses an Arduino as interface between the computer running the controller and the actuator and sensor. Figure 2 shows the full process set-up.

This system was instrumented by Bernardes et al (2006) and was developed to have a wide range of time constants. This makes the control task more challenging. Other interesting feature of this system is the high input delay. Also dead zones and saturation poses challenge to the RL algorithm. Finally, the system has a non-linearity that is square root, and altough it is a smooth non-linearity, also enhances the difficulty of the controll problem.

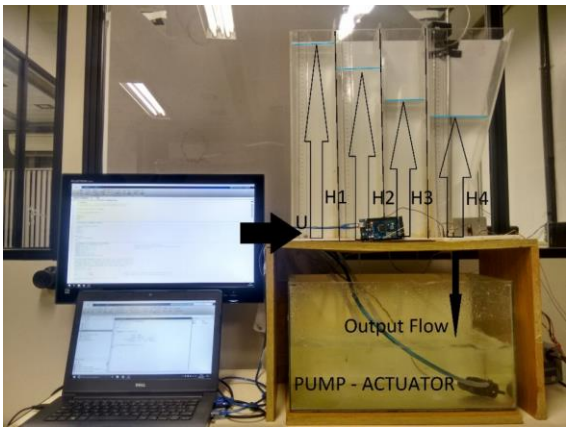

Figure 2. Experimental Set-Up

4 Controller Algorithm Development

4.1 Defining Used Neural Networks

The system dynamic was approximated on the run by a Multilayer Perceptron (MLP) with the backpropagation algorithm in a supervised manner as described in Haykin (2009) and Orr (1996). The objective was to identify a dynamic model of the system in order to predict future outputs. This was necessary because in RL the reward is an immediate response of the environment to an action taken by the agent, but since the system has high valued delay, the reward would not represent instant response of given action, thus, a predictor was used to estimate the future response and then calculate the reward.

There were two functions that needed to be approximated; the value function and the action policy function to be used in the Actor-Critic method. Both were approximated by Radial Basis Neural Networks (RBN) and were trained using backpropagation for the weights, as seen in Wang et al. (2007). The parameter of correction was defined by the TD error. The centers were distributed uniformly through the input space and remained the same throughout the process. The activation function of the RBN was chosen to be the Gaussian function.

From Haykin (2009), the output of the RBN is:

$$y = \sum_{j=1}^{N} w_j \, \rho\left(\left|\left|\boldsymbol{x} - \mu_j\right|\right|\right), \quad (8)$$

where w_j is the weight connected to neuron j, and $\rho\left(\left|\left|\boldsymbol{x} - \mu_j\right|\right|\right)$ is the activation function. The activation function is a radial basis function that depends on the distance to a center parameter μ_j which is the center parameter for neuron j.

As seen in Wang et al. (2007), using the quadratic TD error as the cost function to be minimized, the backpropagation for the weights yields

$$w_j(k+1) = w_j(k) + \eta_a \delta_{td} \rho\left(\left|\left|\boldsymbol{x} - \boldsymbol{c_j}\right|\right|\right), \quad (9)$$

where η_a is the learning rate.

4.2 Calculated Observed Reward

The reward, as said previously in Section 2, is an instant measure of an action's quality. For the control problem, it seems logical to use a function of the output error as the reward function.

Canonically, the environment is the sole responsible to generate a reward, thus, the reward must be a function of the RL state variables. In order to use the output error as parameter for the reward function, either the error itself must be part of the RL state or the output level and the reference signal must together be part of the RL state since the error signal can be calculated from the other two. The RL state variables were chosen to be the reference signal and the output level.

It is important to notice that the RL state variables are not the same as the system's state variables. In the RL problem, the state variables are those enough for the agent to understand in what situation it is. Also, the state is used to generate the pertinent reward to the problem. The chosen information shows where the agent is and where it should go.

The reward function was chosen to be proportional to the error itself, as follows

$$r_k = -|e(k)| \tag{10}$$

where $e(k)$ is the output level error in instant k. The reward is symmetric regarding the y-axis because the RL can receive information on whether the action was good or bad and on *how* good or bad (hence the module function), but it cannot receive information on what to do next. This is why the agent should not know if the output signal is lower of higher than the reference signal. The agent has to decide it's action through trial and error. This is the essence of RL.

4.3 Controller Structure and Algorithm

With the functions for the TD error, the reward function and the updating of the neural networks defined, the iterative algorithm can be developed.

Figure 3 shows the controller structure. Every block is performed by a neural network aside from the system itself which is the real environment. The model identification is performed by a MLP. The actor and critic are performed by a RBN.

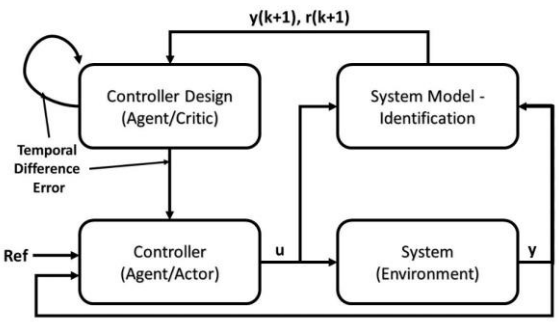

Figure 3. The developed controller structures.

The algorithm used for the controlled is described as follows:

1) Initialize s_k and constants used by the controller.
2) Initialize a_k using the RBN defined for the actor.
3) Initialize Q_k using the RBN defined for the critic.
4) Execute a_k on the model to predict s_{k+1}.
5) Observe r_{k+1}.
6) Calculate a_{k+1} using the RBN defined for the actor.
7) calculate Q_{k+1} using the RBN defined for the critic.
8) 8. Calculate δ_{td}.
9) Update the Neural Networks
10) Optional – Update the system model. (The model can also be trained offline in batch mode)
11) Perform a new measurement to define a new s_k.
12) Calculate a new a_k.
13) Repeat steps 4 to 13.

The algorithm is based on finding the TD error as in Sutton & Barto (1998), but in this case, the TD error is used to update the neural networks. It computes in a sequential way the values needed for calculating the TD error using equation (5) for the SARSA method.

5 Experiments and Results

The first task of this work was to identify a dynamic model of the system in order to predict future outputs. Several network layouts and learning algorithms were tested for batch training with 13 hours of training data tested against 4 hours of validation data. The best results for each case can be seen in Table 1. The multilayer perceptron with two hidden layers was the network that performed better when no previous training was available. Figure 4 shows the network's on-the-run performance with no previous learning. This result is not expected since the multilayer perceptron should be a "slow-learner" yet since it shows the best results, it was the network chosen for this task.

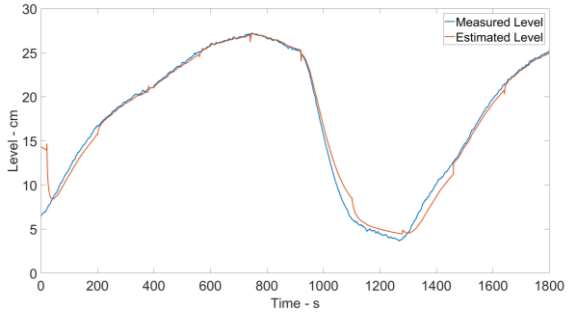

Figure 4. Online identification with multilayer perceptron.

Table 1. Identification Results for the Batch Training

Network Lay-out/Training Algorithm	Num-ber of Neuron in Hid-den Layers	Number of Epochs to Achieve Max Fit	Max Fit
MLP with 2 hidden layers.	90-15	37	92.55%
MLP with 1 hidden layer.	5	100	89.67%
RBN with fixed centers started ran-domly.	30	50	93.05%
RBN with fixed centers started uni-formly.	16	43	96.82%
RBN with ran-domly started centers and Backpropaga-tion.	20	26	93.53%
RBN with Uni-formly started centers and Backpropaga-tion.	9	18	94.35%
RBN with ran-domly started centers using clusters.	30	22	93.88%
RBN with Uni-formly started centers using clusters.	36	50	95.51%
RBN with online creating centers but no center correc-tion.	15	35	67%
RBN with online creating centers with center correc-tion.	15	43	93.28%

The learning rate was varied from 0.001 to 0.05 to test the algorithm's stability margins regarding learning rate. Figure 5 shows some results of these tests. Each graphic has, respectively, learning rate of 0.005, 0.01, 0.025 and 0.05. Since the learning rate is analogous to a step size, a very small step does not bring the controller to its goal in feasible time while a very large step renders the controller unstable. The graphics in Figure 5 were obtained by simulation. Figure 6 shows a real environment performance with the same conditions as the third graphic in Figure 5. This indicates that the real environment's behavior is alike the simulated behavior so most of the inferences made from the simulated tests can be used for the controller tuning.

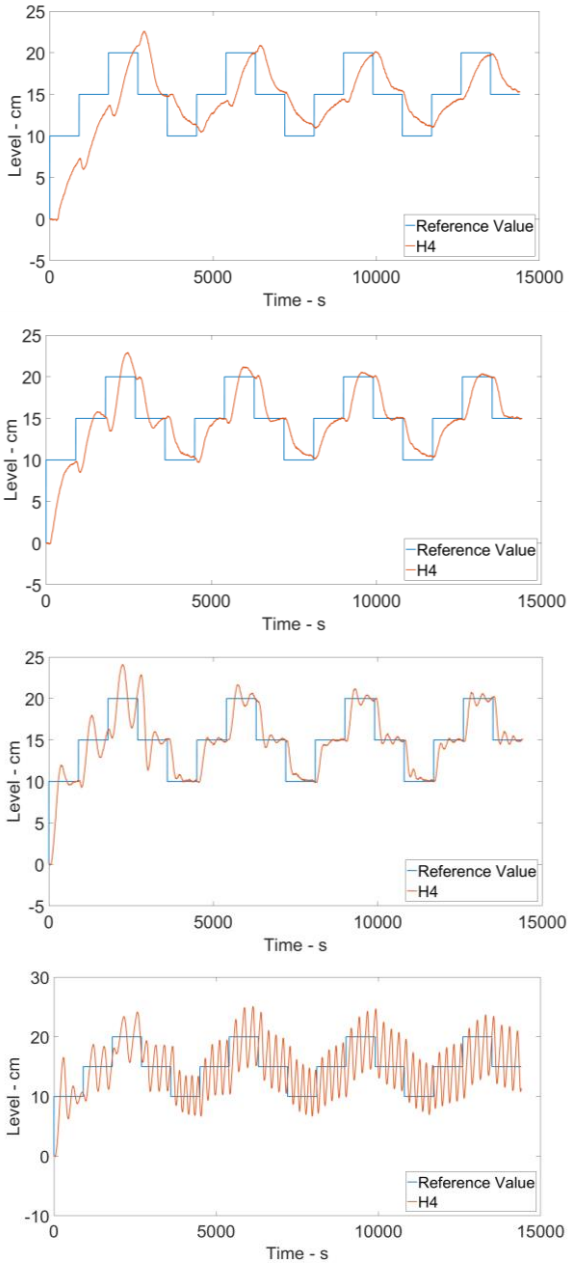

Figure 5. Effect of different learning rates.

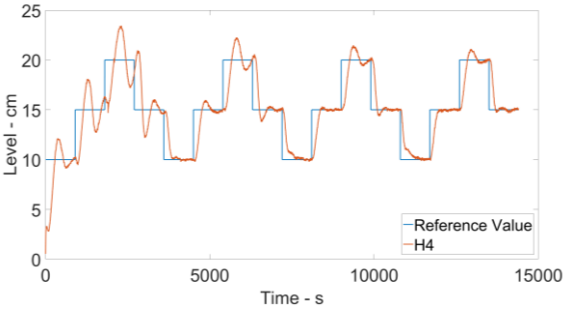

Figure 6. Real environment test. Learning rate of 0.025. Compare to Graphic 3 in Figure 5.

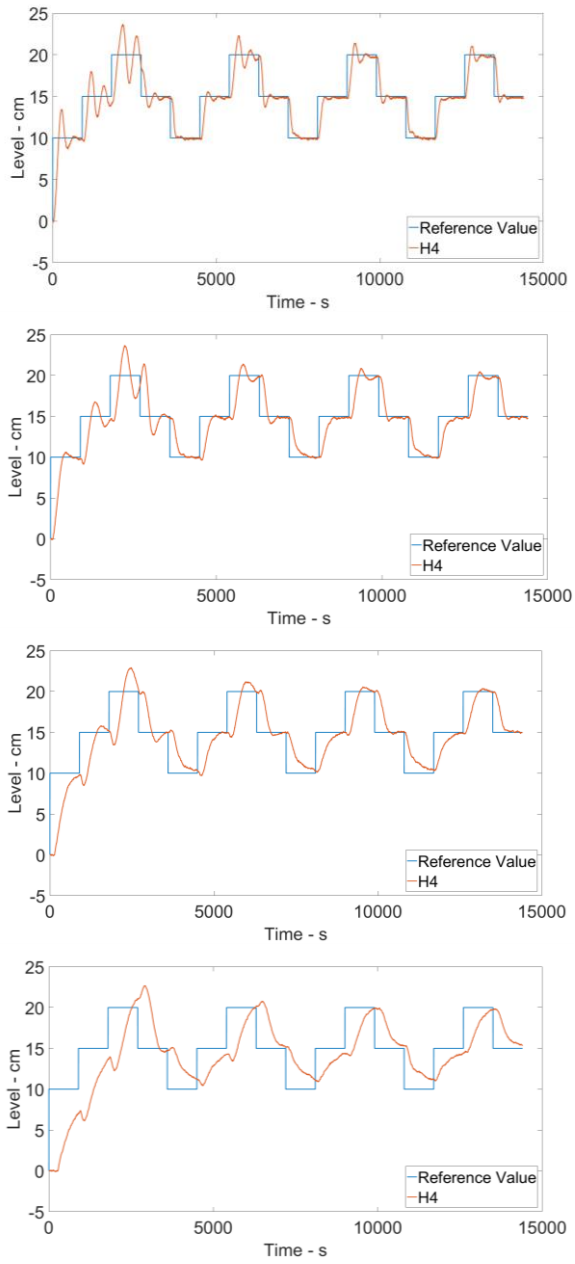

Figure 7. Effect of different sample times.

Sample rate was also evaluated. The learning rate is analogous to a step size and the sample rate is analogous to number of steps, so a correlation is in order. The sample time was varied from 0.1s to 2s and some of the results can be seen in Figure 7. Each graphic has, respectively, sample time of 0.1s, 0.5s, 1s and 2s. A smaller sample rate will make the controller slower and unable to reach its goal in feasible time while a larger sample rate might make the controller unstable or with too much oscillation. The challenge lies in finding a good learning rate/sample time couple.

The chosen sample time for the controller to be tested against a comparison was 0.5s and the learning rate was chosen to be of 0.008. The controller performed against a PI projected to have least overshot in the reference of 15cm. Figure 8 shows the RL controller's performance against that of the PI controller and Figure 9 shows the respective control signals. In the beginning of the test, the RL controller performs poorly if compared to the PI because it has no information on the system, but given time, the developed controller shows adaptive and optimizing features. This causes the performance to enhance over time and, for most reference points, reach it faster than the PI and with least oscillation.

Figure 10 is a simulated result that shows what happens when the controller is given enough time to learn. It can be seen that the controller learns to lower overshoot and to stabilize around the reference value faster. From Figures 5 and 6 it can be inferred that this behavior will be reproduced by the real environment controller.

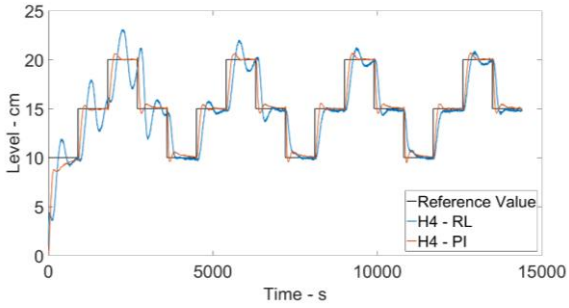

Figure 8. Comparison against a PI for the test references.

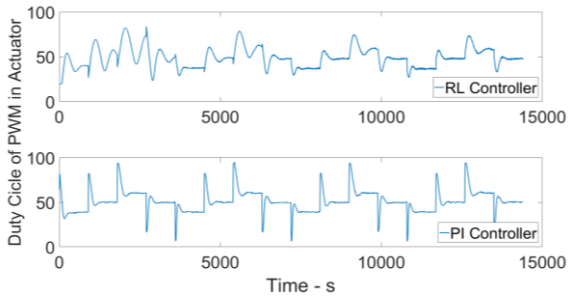

Figure 9. Control signal of the controllers for the first comparison.

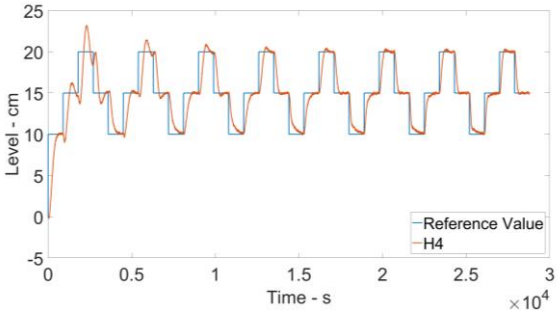

Figure 10. Performance overtime.

Figure 11 shows the comparison between the developed controller against the same PI from previous comparison for random reference signals and Figure 12 shows the respective control signals. The PI, in some cases, never reaches the reference value and when it does, is sometimes outperformed by the RL controller. Also, the RL controller tends to enhance performance over time lowering overshoot for all cases while the PI's behavior will remain the same.

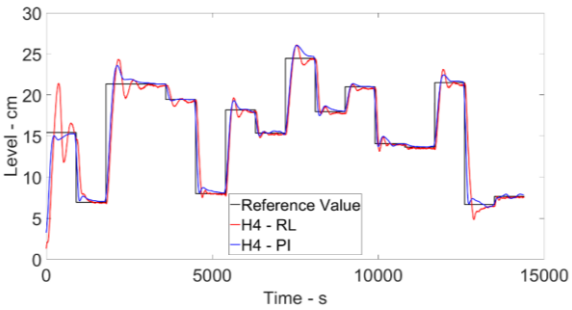

Figure 11. Comparison against a PI for random references.

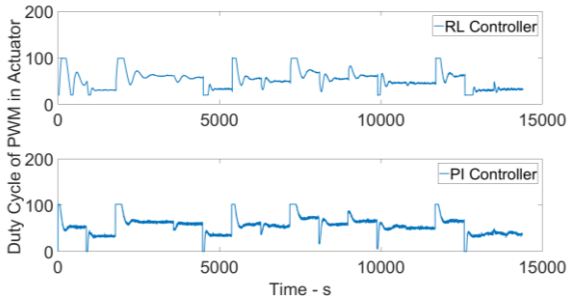

Figure 12. The developed controller structure.

6 Conclusion

This work presented an adaptive controller based on reinforcement learning using neural networks. The goal of this work was to tune a RL controller for usage in a real environment. The controller was able to perform when the margins and correlation of learning rate and sample rate were respected. It showed learning and over time improvement features. With its adaptive and optimizing features was able to perform better then the PI used for comparison if given time.

Important future work lies in testing the same controller in different systems, with different order and dynamics, changing learning and sample rate, to evaluate how robust is the algorithm, also, it is important to test how the controller deals with time variance, change the valves positions online.

References

Aguirre, L. A. (2015). Introdução à Identificação de Sistemas, Quarta Edição. Editora UFMG.

Bernardes, M. C., Borges, G. A, G. A. F. Melo, A. A. Freitas, G. A. Borges, and A. Bauchspiess (2006). Instrumentação e estimação de parâmetros de um sistema de nível de líquidos de quatro tanques interligados. *XII Congresso Brasileiro de Automática*, Salvador 2006. pp. 3427-3432.

Cui, R., C. Yang, Y. Li, and S. Sharma (2017, jun). Adaptive neural network control of AUVs with control input nonlinearities using reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems.* 47(6), 1019–1029.

Esfe, M. H. (2017). Designing an artificial neural network using radial basis function (rbf-ann) to model thermal conductivity of ethylene glycolâwater-based tio2 nanofluids. J Therm anal Calorim.

Ferrari, S. and R. F. Stengel (2005). Smooth function approximation using neural networks. IEEE Transactions on Neural Networks, VOL. 16, NO. 1, January 2005.

Haykin, S. S. (2009). *Neural Networks and Learning Machines.* Prentice Hall.

Hwangbo, J., I. Sa, R. Siegwart, and M. Hutter (2017, oct). Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters 2(4), 2096–2103.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D.Wierstra (2016). Continuous control with deep reinforcement learning. ICLR.

Noel, M. M. and B. J. Pandian (2014, oct). Control of a nonlinear liquid level system using a new artificial neural network-based reinforcement learning approach. *Applied Soft Computing* 23, 444–451.

Orr, M. J. L. (1996). Introduction to radial basis function networks.

Papierok, S., A. Noglik, and J. Pauli (2008). Application of reinforcement learning in a real environment using an RBF network. *1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems.*

Ramanathan, P., K. K. Mangla, and S. Satpathy (2018, feb). Smart controller for conical tank system using reinforcement learning algorithm. *Measurement* 116, 422–428.

Rao, J., H. An, T. Zhang, Y. Chen, and H. Ma (2016, aug). Single leg operational space control of quadruped robot based on reinforcement learning. In 2016 IEEE *Chinese Guidance, Navigation and Control Conference* (CGNCC). IEEE.

Sutton, R. S. and A. G. Barto (1998). Reinforcement Learning: An Introduction. MIT Press.

Sutton, R., A. Barto, and R. J. Williams (1991). Reinforcement learning is direct adaptive optimal control. *American Control Conference.*

Vamvoudakis, K. G., M. F. Miranda, and J. P. Hespanha (2016, nov). Asymptotically stable adaptive–optimal control algorithm with saturating actuators and relaxed persistence of excitation. *IEEE Transactions on Neural Networks and Learning Systems* 27(11), 2386–2398.

Vrabie, D., Vamvoudakis, K. G. And Lewis, F. L., Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles. *The Institution of Engineering and Technology, London.*

Wang, X. S., Y. H. Cheng, and W. Sun (2007, mar). A proposal of adaptive PID controller based on reinforcement learning. *Journal of China University of Mining and Technology* 17(1), 40–44.

Yu, H., P. D. Reiner, T. Xie, T. Bartczak, and B. M. Wilamowski (2014). An incremental design of radial basis function networks. *IEEE Transactions on Neural Networks and Learning Systems*, VOL. 25, NO. 10, October 2014.